What you really need to know about recurrences

- Work per level changes geometrically with the level
- Geometrically increasing ($x > 1$)
  - The bottom level wins
- Geometrically decreasing ($x < 1$)
  - The top level wins
- Balanced ($x = 1$)
  - Equal contribution

### T(n) = aT(n/b) + n^c

- Balanced: $a = b^c$
  - $T(n) = 16T(n/4) + n^2$
- Increasing: $a > b^c$
  - $T(n) = 5T(n/3) + n$
  - $T(n) = 3T(n/4) + n^{1/2}$
- Decreasing: $a < b^c$
  - $T(n) = T(4n/5) + n$
  - $T(n) = 7T(n/2) + n^3$

### Divide and Conquer Algorithms

- Split into sub problems
- Recursively solve the problem
- Combine solutions
- Make progress in the split and combine stages
  - Quicksort – progress made at the split step
  - Mergesort – progress made at the combine step
- D&C Algorithms
  - Strassen’s Algorithm – Matrix Multiplication
  - Inversions
  - Median
  - Closest Pair
  - Integer Multiplication

### How to multiply 2 x 2 matrices with 7 multiplications

<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
<th>a</th>
<th>b</th>
<th>e</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>u</td>
<td>c</td>
<td>d</td>
<td>f</td>
<td>h</td>
</tr>
</tbody>
</table>

- $r = p_1 + p_2 - p_4 + p_6$
- $s = p_4 + p_5$
- $t = p_6 + p_7$
- $u = p_2 - p_3 + p_5 - p_7$

Where:

- $p_1 = (b - d)(f + h)$
- $p_2 = (a + d)(e + h)$
- $p_3 = (a - c)(e + g)$
- $p_4 = (a + b)h$
- $p_5 = a(g - h)$
- $p_6 = d(f - e)$
- $p_7 = (c + d)e$

Aho, Hopcroft, Ullman 1974

### Strassen’s Algorithms

- Treat $n \times n$ matrices as $2 \times 2$ matrices of $n/2 \times n/2$ submatrices
- Use Strassen’s trick to multiply $2 \times 2$ matrices with 7 multiplies
- Base case standard multiplication for single entries
- Recurrence: $T(n) = 7T(n/2) + cn^2$
- Solution is $O(7^{\log_2 n}) = O(n^{\log_7 7})$ which is about $O(n^{2.807})$
Inversion Problem
- Let \( a_1, \ldots, a_n \) be a permutation of 1 \( \ldots \) n
- \((a_i, a_j)\) is an inversion if \( i < j \) and \( a_i > a_j \)

\[ 4, 6, 1, 7, 3, 2, 5 \]

- Problem: given a permutation, count the number of inversions
- This can be done easily in \( O(n^2) \) time
  - Can we do better?

Application
- Counting inversions can be used to measure how close ranked preferences are
  - People rank 20 movies, based on their rankings you cluster people who like that same type of movie

Counting Inversions

\[
\begin{array}{cccccccccccccc}
11 & 12 & 4 & 1 & 7 & 2 & 3 & 15 & 9 & 5 & 16 & 8 & 6 & 13 & 10 & 14 \\
\end{array}
\]

Count inversions on lower half
Count inversions on upper half
Count the inversions between the halves

Count the Inversions

\[
\begin{array}{cccccccccccccc}
11 & 12 & 4 & 1 & 7 & 2 & 3 & 15 & 9 & 5 & 16 & 8 & 6 & 13 & 10 & 14 \\
5 & 2 & 1 & 10 & 9 & 11 & 14 & 16 & 6 & 8 & 13 & 15 & 12 & 7 & 3 \\
\end{array}
\]

Problem – how do we count inversions between sub problems in \( O(n) \) time?
- Solution – Count inversions while merging

\[
\begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 7 & 11 & 12 & 15 & 5 & 6 & 8 & 9 & 10 & 13 & 14 & 16 \\
\end{array}
\]

Standard merge algorithm – add to inversion count when an element is moved from the upper array to the solution

Use the merge algorithm to count inversions

\[
\begin{array}{cccccccccccccc}
1 & 4 & 11 & 12 & 2 & 3 & 7 & 15 \\
\end{array}
\]

\[
\begin{array}{cccccccccccccc}
5 & 6 & 8 & 16 & 6 & 10 & 13 & 14 \\
\end{array}
\]

Indicate the number of inversions for each element detected when merging.
Inversions

- Counting inversions between two sorted lists
  - O(1) per element to count inversions

![Inversion Example]

- Algorithm summary
  - Satisfies the “Standard recurrence”
  - T(n) = 2 T(n/2) + cn

Computing the Median

- Given n numbers, find the number of rank n/2
- One approach is sorting
  - Sort the elements, and choose the middle one
  - Can you do better?

Problem generalization

- Selection, given n numbers and an integer k, find the k-th largest

Select(A, k)

```plaintext
Select(A, k)
    Choose element x from A
    S_1 = {y in A | y < x}
    S_2 = {y in A | y > x}
    S_3 = {y in A | y = x}
    if (|S_2| >= k)
        return Select(S_2, k)
    else if (|S_2| + |S_3| >= k)
        return x
    else
        return Select(S_1, k - |S_2| - |S_3|)
```

Randomized Selection

- Choose the element at random
- Analysis can show that the algorithm has expected run time O(n)

Deterministic Selection

- What is the run time of select if we can guarantee that choose finds an x such that |S_1| < 3n/4 and |S_2| < 3n/4 in O(n) time
BFPRT Algorithm

- A very clever choose algorithm . . .

Split into \( n/5 \) sets of size 5
M be the set of medians of these sets
Let x be the median of M

BFPRT runtime

\(|S_1| < 3n/4, |S_2| < 3n/4\)

Split into \( n/5 \) sets of size 5
M be the set of medians of these sets
x be the median of M
Construct \( S_1 \) and \( S_2 \)
Recursive call in \( S_1 \) or \( S_2 \)

BFPRT Recurrence

- \( T(n) \leq T(3n/4) + T(n/5) + c \ n \)

Prove that \( T(n) \leq 20 \ c \ n \)